DyQ-VLA: DYNAMIC QUANTIZATION AND COMPENSATION FOR VLA MODELS WITH RUNTIME ERROR AWARENESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language-action (VLA) models achieve favorable task performance, yet runtime errors in closed-loop execution evolve with alternating updates of actions and observations. Existing VLA quantization methods mainly trade off inference speed and model performance while rarely investigating how quantization alters runtime errors. By comparing error distributions between full-precision and quantized models, we identify distinct evolutionary patterns for the two types of errors: quantization enlarges the variance of translation error distributions and increases their dispersion, whereas the distribution center of rotation errors gradually shifts across execution steps, demonstrating cumulative drift. Motivated by this observation, we rethink the optimal quantization strategy for VLA models and propose *DyQ-VLA*, a runtime-error-aware quantization framework, which dynamically selects activation precision according to execution steps and tracks as well as compensates rotation errors via accumulated quantization residuals. Corresponding operators and runtime adaptation strategies are devised within the framework to enable dynamic-precision execution. Experiments show that *DyQ-VLA* achieves a 1.88× to 1.93× inference speedups while maintaining comparable or higher average task success rates. Moreover, it reduces mean execution steps by 17.7% to 19.8%. Our code is here: https://anonymous.4open.science/r/DyQ-VLA-7F51/.

1 INTRODUCTION

Vision-Language-Action (VLA) models unify visual perception, language understanding, and action generation within a single framework (Brohan et al., 2023; Kim et al., 2024). Their strong performance across diverse robotic manipulation tasks has established them as a mainstream approach to embodied intelligence (Physical Intelligence et al., 2025; Zhang et al., 2025; Din et al., 2025).

In practical VLA execution, deviations between actual and expected trajectories (e.g., human-centric demonstration trajectories) give rise to **runtime errors**, as shown in Fig. 1 (a). A model may still complete the task by correcting these errors, but repeated corrections increase the number of inference steps and task completion time. As current errors affect subsequent observations and actions, they evolve into a dynamic **action-error correlation**. Existing methods, such as FLARE (Zhao et al., 2026) and REPO-VLA (Liufu et al., 2026), seek to correct execution errors online, yet this correlation remains difficult to characterize explicitly.

Meanwhile, quantization reduces VLA memory use and inference latency, becoming a popular optimization methods for VLA real-time application (Xu et al., 2026c; Zhang et al., 2026a). Existing approaches (e.g. QVLA (Xu et al., 2026c), QuantVLA (Zhang et al., 2026a) and ActQuant (Akbari et al., 2026)) typically assess whether faster inference preserves task success, but give less attention to how quantization changes the action-error correlation and the corrections it requires. We therefore ask: ***How does quantization alter action-error correlation, and how can this understanding guide a quantization strategy that reduces overall task completion time?***

In this paper, we first analyze how runtime errors evolve and how quantization changes their relationship with subsequent actions. Decomposing action errors into translation and rotation reveals distinct patterns: translational errors vary substantially around a relatively stable mean, while rotational errors persist more consistently across steps. Quantization further disperses the translational error

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

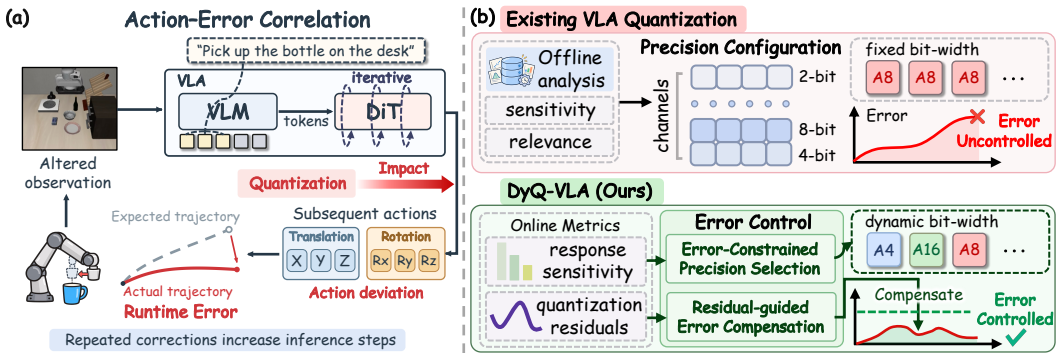

Figure 1: **Action-Error Correlation and *DyQ-VLA* Overview.** (a) Runtime errors and actions interaction of VLA Models. (b)Runtime-Error-Aware VLA Quantization Framework – *DyQ-VLA*.

distribution but only slightly alters the rotational error distribution. To characterize these effects at runtime, we introduce metrics for the action response to existing errors and for the accumulated rotational error. These findings motivate two key insights: dynamic activation precision to control translational error dispersion and residual compensation to mitigate rotational drift.

Based on the foregoing analysis, we propose *DyQ-VLA*, a runtime-error-aware dynamic quantization framework for VLA models. This framework incorporates two core mechanisms: (1) dynamic selection of activated quantization precision at each inference step with static weight precision preserved to effectively regulate translation error distribution; (2) continuous residual-based compensation for rotation error distribution during execution for effective rotation error control. Moreover, we implement runtime adaptations for *DyQ-VLA*, including dynamically precise quantization operators, efficient operator fusion strategies, and system-level scheduling optimization.

In summary, our contributions are below:

- We explicitly analyze action-error correlation, identify error distribution characteristics of its different components, and reveal quantization's effect on error distributions, offering key insights to rethink VLA quantization strategies.
- Based on these observations, we rethink the optimal quantization strategy for VLA models and propose *DyQ-VLA*, the first runtime-error-aware VLA quantization framework to our knowledge. *DyQ-VLA* integrates a dynamic activation quantization strategy to mitigate translational error components and a residual-based compensation mechanism to suppress rotational components, with runtime optimization and adaptation.
- We perform simulation and real-world tests on *DyQ-VLA* and show that it outperforms SOTA methods in runtime error control and inference speed, which further validates that VLA quantization should be investigated from the perspective of runtime error.

On popular simulation benchmarks and real-world scenarios, *DyQ-VLA* achieves up to 1.93× speedups and reduces mean translation and rotation runtime errors to 78.5% - 87.0% and 59.4% - 82.5%, respectively. Moreover, it reduces mean execution steps by 17.7% to 19.8%.

2 Background

2.1 Runtime Error in VLA Models

Runtime Error Between Execution and Expected in VLA Inference. During VLA execution, runtime error denotes the deviation of actual trajectories from desired ones such as human demonstration trajectories. Due to model robustness and environmental fault tolerance, such errors do not necessarily lead to task failure, yet extra planning for error correction increases inference steps and task completion time (Wang et al., 2026a;b). Accordingly, VLA acceleration demands both lower per-inference cost (Xu et al., 2026a) and fewer error-triggered extra inferences (Wang et al., 2026b; Li et al., 2025), constituting two complementary optimization directions.

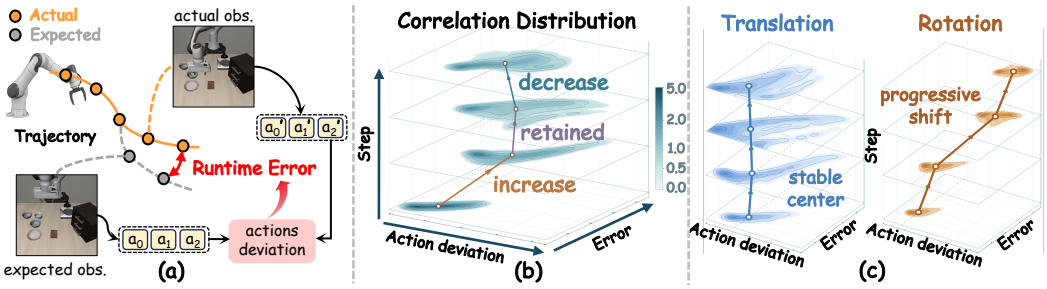

Figure 2: **Action–Error Correlation in Closed-Loop VLA Execution.** (a) Runtime errors alter observations and subsequent actions, which in turn change runtime errors. (b) Joint distributions of correlation changes in both center and shape across steps. (c) The distribution for translation remains dispersed around a stable center, while rotation stays concentrated and shifts toward larger errors.

Action-Error Correlation in VLA Models. In VLA models, runtime errors interfere with subsequent action generation covering translation (X, Y, Z), rotation (R_X, R_Y, R_Z) and gripper control, while action execution alters future errors and forms a dynamically coupled action-error correlation. Current research targets runtime error analysis and mitigation via learning-based methods such as FLARE (Zhao et al., 2026) and REPO-VLA (Liufu et al., 2026), which lack explicit characterisation of action-error feedback relationships and remain under investigation regarding their stepwise evolutionary mechanisms.

2.2 VLA Model Quantization

Precision Selection and Accuracy-Speed Trade-off for VLA Model Quantization. Quantization cuts storage and computational costs and speeds up inference by decreasing the numerical precision of model parameters, acting as an effective optimization strategy for VLA models. Existing research on VLA model quantization mainly explores precision selection and the accuracy-speed tradeoff, such as QVLA (Xu et al., 2026c), DA-PTQ (Xu et al., 2026b), and ActQuant (Akbari et al., 2026). These methods achieve certain effects for VLA real-time deployment.

Quantization Impact on Action-Error Correlation in VLA Models. However, quantization performs low-precision transformations on parameters, alters the distribution of internal representations, distorts the model's input-output mapping, and affects runtime errors (Zhang et al., 2026a). Thus, quantization raises the complexity of the relationship between errors and actions (Wang et al., 2026c), which prevents the direct application of original correlation-based error analysis and compensation to quantized models. Current works cannot analyze quantization's effects on runtime errors and overlook extended inference steps induced by runtime errors, leading to suboptimal speedup (Sanchez et al., 2026; Kim et al., 2026). Additional related work is provided in Appendix A.

3 Runtime Error Analysis for VLA Model Quantization

3.1 Analysis of Action-Error Correlation in Full-Precision VLA

Complexity of Action-Error Correlation. During closed-loop VLA execution, runtime errors alter subsequent input observations and thereby affect subsequent action predictions. Executing these actions in turn changes runtime errors, forming a complex coupled action–error correlation. As shown in Fig. 2 (a), the error at step t alters the observation and predicted action sequence, resulting in a larger error at $t + H$. The current runtime error thus depends on both the current action and historical errors. Such cross-step dependence complicates the analysis and modeling of runtime errors.

Joint Distribution for Modeling Action-Error Correlation. To model action-error correlation, we investigate the joint distribution of action deviations and runtime errors over steps via repeated rollouts with equal-magnitude perturbations at a fixed execution step. As illustrated in Fig. 2 (b), the centroid of the correlation distribution varies across steps and presents error amplification, attenuation or relative retention. Its width also expands or contracts with steps, and variations in both centroid and width characterize the complexity of this correlation distribution.

Error Decomposition in terms of Translation/Rotation Distribution. Due to complex variations in the above distribution, further analysis is challenging. To reveal its inherent rules, we decompose actions into translational and rotational components and separately analyse their action-error correlation distributions. As illustrated in Fig. 2 (c), both correlation distributions concentrate on small values initially; the rotational distribution is relatively compact, while the translational one is more dispersed. As execution continues, the rotational distribution stays concentrated, and its mean gradually shifts toward larger errors in a consistent direction. By contrast, the translational distribution maintains dispersion around a stable mean throughout all steps. These observations demonstrate divergent evolution patterns of action-error correlation for different motion components.

***Observation* ①:** *Action-Error Correlation exhibits distinct component-wise patterns: the translational distribution has high variance and a relatively stable central value, while the rotational distribution has low variance and a center that progressively shifts in the same direction toward larger errors.*

3.2 Quantization Impact on Action-Error Correlation in VLA Models

Sensitivity of Translation/Rotation Distribution to Quantization. However, quantization will further affects aforementioned correlation. To investigate the impact of quantization on such correlation, we group samples by action direction and compare error distributions before and after quantization at each step. We define sensitivity as the change in subsequent action predictions per unit runtime error as shown in Eq. (1). Here, $\boldsymbol{A}^{(j)}$ and $\overline{\boldsymbol{A}}$ denote predictions from matched perturbed and unperturbed executions at the same precision, h is the prediction horizon, and $\varepsilon_t^{(j)}$ is the normalized error magnitude in direction j. Differences in sensitivity before and after quantization measure its impact on action-error correlation.

$$\mathscr{S}_{t,j} = \frac{1}{\varepsilon_t^{(j)}} \sqrt{\frac{1}{h} \sum_{i=1}^{h} \left\| \boldsymbol{A}_{t+1,i}^{(j)} - \overline{\boldsymbol{A}}_{t+1,i} \right\|_2^2}. \tag{1}$$

Quantization Impact on the Translation Error Distribution. Fig. 3 (a) shows that quantization substantially alters the translation error distribution and increases its variance. Quantization further amplifies some large errors into more pronounced outliers and shifts the distribution center toward larger errors. Sensitivity analysis further reveals a stronger action response per unit translation error after quantization, with sensitivity varying across steps. This indicates that quantization significantly alters the translation action-error correlation, increasing error variance and shifting the distribution center toward larger errors, with varying effects on sensitivity across steps.

Quantization Impact on the Rotation Error Distribution. Fig. 3 (b) shows that under identical conditions, the overall distribution pattern of rotation errors changes slightly before and after quantization, with weaker amplification of extreme errors than in translation. Using the same sensitivity metric, we find that the action response per unit rotation error changes only slightly after quantization, with little fluctuation across execution steps. This indicates that quantization has a limited effect on the rotation action-error correlation, with smaller distribution changes and weaker single-step response amplification than in translation.

***Observation* ②:** *Quantization increases translation error variance and response sensitivity, with its impact on sensitivity varying across steps. In contrast, rotation error distributions and response sensitivity change only slightly after quantization.*

3.3 Residual Metric for Quantization Impact on Action-Error Correlation

Quantization Residuals for Tracking Rotation Errors. To support compensation for progressive shifts in the rotation error distribution, we project the difference between activations and their quantized reconstructions onto rotational action components. We accumulate these signed projections across steps in a common coordinate frame to construct a cumulative quantization residual metric:

$$\boldsymbol{\mathcal{R}}_t^R = \sum_{k=0}^{t-1} \boldsymbol{\zeta}_k^R, \qquad \left\| \boldsymbol{\mathcal{R}}_t^R \right\|_2^2 = \sum_{k=0}^{t-1} \left\| \boldsymbol{\zeta}_k^R \right\|_2^2 + 2 \sum_{0 \le i < j < t} \langle \boldsymbol{\zeta}_i^R, \boldsymbol{\zeta}_j^R \rangle. \tag{2}$$

Here, $\boldsymbol{\zeta}_k^R$ denotes the signed rotational projection of the activation quantization residual at query k. Positive and negative cross terms capture reinforcement and cancellation across steps, respectively.

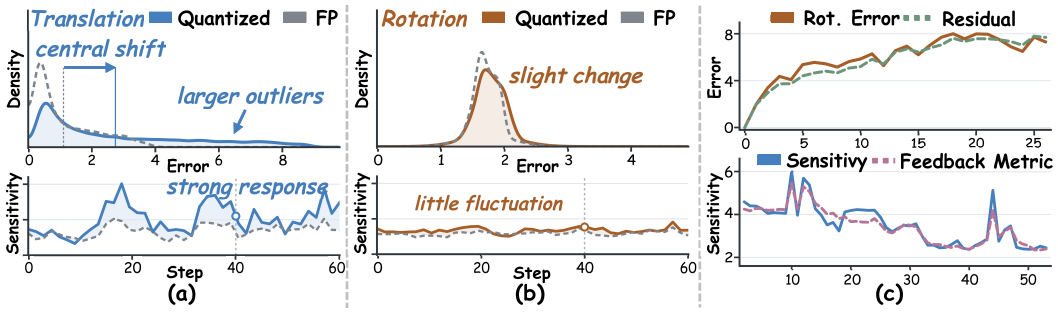

Figure 3: **Quantization Effects and Online Tracking.** (a) Quantization increases translation error variance and response sensitivity. (b) Rotation error distributions and sensitivity change only slightly. (c) Residual and feedback metrics track rotational error and translational sensitivity, respectively.

By preserving historical direction information, this metric provides an online proxy for the current net residual of rotation errors.

Execution Feedback for Characterizing Response Sensitivity. To constrain the single-step amplification of translation errors, we estimate the response sensitivity defined in Sec. 3.2 using deviations between actual and planned translation, together with changes in rotational actions. At each step, this estimate characterizes the change in subsequent action predictions per unit translation error, providing an online signal for constraining extreme action responses.

Online Metrics for Guiding Quantization Control. Fig. 3 (c) presents the tracking results for both metrics. The cumulative quantization residual closely follows rotation runtime error measured against the matched reference trajectory, while the feedback metric tracks changes in measured translation response sensitivity across steps. Accordingly, cumulative residuals can guide compensation direction and magnitude to suppress continuous drift in the rotation error distribution. The estimated translation response sensitivity guides per-step deviation bounds and precision selection within these bounds, helping limit further amplification of extreme translation errors. We extend the above analyses across multiple scenarios and observe consistent patterns, supporting the generality (Appendix. C).

Key Insights for VLA Quantization Design: *The proposed metrics characterize and track quantization effects on both distributions. We should use these metrics to guide runtime dynamic precision adjustment for translation error control and effective rotation error compensation.*

4 Runtime Error Aware Dynamic Quantization Framework

4.1 Dynamic Quantization Strategy for Translation Error Control

Offline Error and Latency Profiling. Aformentioned analysis reveals that translation error distribution at each step can be controlled by regulating quantization precision, while rotation error distribution can be corrected via continuous compensation. First, we explore that precision selection requires error and latency estimates for candidate configurations. Since quantization loss mainly arises from activations (Lin et al., 2024b), we retain static weight quantization and profile activation precision configurations offline, avoiding per-candidate evaluation at runtime. Sepcifically, we estimate $E_d(b)$ using high quantiles of translational and rotational errors and record inference latency $L(b)$. The reference configuration provides the calibration error baseline τ_d.

Runtime-Error-Aware Quantization Precision Selection. Our analysis shows that quantization increases translation error variance, making the distribution more dispersed and harder to track. Existing studies overlook this issue and use static activation quantization. We therefore dynamically select activation precision to constrain this variance at each step. Before inference t, we estimate the normalized response sensitivity $\widehat{s}_{t,d}$ from the previous execution's translation shortfall relative to its plan and rotation variations in the existing motion plan. Using offline profiles, we select the configuration minimizing inference latency under translation and rotation error constraints:

$$\arg\min_{b\in\mathcal{B}} L(b) \quad \text{s.t.} \quad \max_{d\in\{T,R\}} \frac{\widehat{s}_{t,d}\big(E_d(b) + r_{t-1,d}\big)}{\tau_d} \le 1. \tag{3}$$

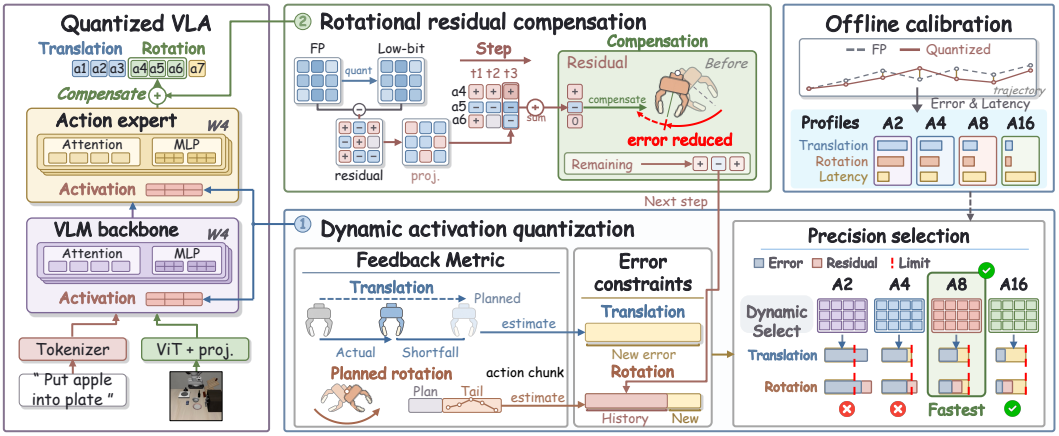

Figure 4: **Framework of *DyQ-VLA*.** Runtime feedback guides dynamic activation quantization for translation error control, while cumulative residuals guide rotation error compensation.

Here, $\mathcal{B}$ is the set of candidate activation precision configurations, and T and R denote translation and rotation, respectively. To account for persistent historical errors, $r_{t-1,R}$ denotes the estimated magnitude of the projected rotation residual remaining after the previous compensation, on the same scale as $E_R(b)$; $r_{t-1,T} = 0$. Higher sensitivity or larger residuals tighten the error bound. Selection uses table lookup and constraint checks, with a predefined fallback if no candidate is feasible. Together, these mechanisms mitigate quantization effects on the translation error distribution while reducing unnecessary high-precision computation.

4.2 Residual Compensation Strategy for Rotation Error Control

Historical Rotation Residual Estimation. Our analysis shows that the rotation error distribution is concentrated and little affected by quantization, yet its center continuously drifts in one direction, causing large accumulated errors late in execution. Existing methods apply global compensation, overlooking differences in distribution evolution across motion components and risking under- or over-compensation. To determine the correction direction and magnitude, we estimate the current net residual of historical rotation errors using the cumulative quantization residual metric in Sec. 3.3.

Residual-Guided Action Compensation. Using this signed residual estimate, we apply corrections in the opposite direction to counteract the drift. We express the rotational residual estimate $\widehat{\mathbf{e}}_t$ and rotation command $\mathbf{a}_{t,i}$ in a common coordinate system and normalized scale. To respect action bounds, we limit the compensation ratio and distribute the correction across the h actions executed in the current round, as shown in Eq. (4). Here, $\eta \in (0, 1]$ controls compensation strength, and absolute values and inequalities are evaluated componentwise. To avoid repeated compensation, we update the historical residual estimate using the actual rotation command corrections. The remaining estimated residual guides the next round of precision selection and action compensation. Together, residual estimation and bounded action compensation suppress persistent drift of the rotation error distribution, keeping its center at a low error level.

$$\alpha_t = \eta \max_{0 \le \alpha \le 1} \left\{ \alpha : |\mathbf{a}_{t,i}| + \frac{\alpha}{h} |\widehat{\mathbf{e}}_t| \le \mathbf{1},\ \forall i \right\}, \qquad \widetilde{\mathbf{a}}_{t,i} = \mathbf{a}_{t,i} - \frac{\alpha_t}{h} \widehat{\mathbf{e}}_t, \quad i = 1, \ldots, h. \tag{4}$$

4.3 Runtime Optimization for *DyQ-VLA* Framework

Dynamic Quantization Kernels with Shared Weights. Dynamic activation precision with fixed W4 weights requires matching different Tensor Core operand formats while preserving shared weight storage. We implement dedicated A2/A4/A8/A16 kernels: A2 uses structured sparse INT4 computation, while A4 uses dense INT4 computation. The A8 path expands W4 fragments to INT8 in registers, and A16 reconstructs W4 tiles as BF16 in shared memory. All paths access the same resident W4 allocation through zero-copy views, keeping format conversions on chip and avoiding additional global weight copies or repacking.

Table 1: **Quantization Results on the LIBERO Simulation Benchmark.**

Method	$\pi_{0.5}$								GR00T-N1.5							
	Success rate (%) ↑					Steps↓	Error ↓		Success rate (%) ↑					Steps↓	Error ↓	
	Spa.	Obj.	Goal	Long	**Avg.**		Trans.	Rot.	Spa.	Obj.	Goal	Long	**Avg.**		Trans.	Rot.
BF16 ref.	98.4	97.8	98.4	93.4	97.0	159.7	60.37	14.19	87.4	91.8	89.4	75.8	86.1	237.2	85.40	18.35
W4A16																
GPTQ	15.0	39.0	10.0	0.0	16.0	248.0	161.19	27.62	43.0	73.0	49.0	60.0	56.3	320.0	160.32	26.91
AWQ	4.0	29.0	11.0	2.0	11.5	255.0	171.22	29.26	38.0	74.0	51.0	64.0	56.8	322.0	167.18	28.03
OmniQuant	8.0	23.0	10.0	0.0	10.3	260.0	176.23	29.93	39.0	65.0	43.0	53.0	50.0	338.0	176.80	29.52
ActQuant	**98.8**	99.0	96.4	91.6	96.5	191.2	72.08	15.52	87.8	93.0	86.8	73.8	85.4	275.1	102.56	19.65
QVLA	98.0	97.2	96.4	91.8	95.9	193.0	72.58	14.56	86.0	90.4	86.6	72.6	83.9	272.5	106.46	20.35
W4A8																
SmoothQuant	98.2	98.4	95.6	93.4	96.4	169.6	72.50	18.81	82.6	91.6	88.2	78.8	85.3	255.4	103.16	22.31
DuQuant	94.6	98.0	93.8	91.0	94.4	168.2	74.94	19.06	73.4	86.6	81.2	77.8	79.8	263.0	113.46	23.05
QuantVLA	98.7	98.0	**98.0**	**96.0**	97.7	186.4	70.33	18.17	**93.6**	93.2	**90.4**	73.4	87.7	267.2	97.77	21.86
W4A4																
SmoothQuant	83.0	88.0	40.0	26.0	59.3	211.0	139.02	28.22	85.2	84.8	81.6	79.4	82.8	259.0	147.25	28.80
DuQuant	96.2	98.2	93.4	88.6	94.1	171.5	91.47	20.75	64.4	74.8	63.2	79.0	70.4	284.0	148.33	28.89
QuantVLA	96.0	**100.0**	**98.0**	86.0	95.0	168.6	85.19	19.88	63.2	71.8	60.8	67.4	65.8	269.5	154.74	29.81
HoloQ-VLA	97.0	**100.0**	96.0	94.0	96.8	189.6	84.72	16.64	86.8	90.8	90.0	79.2	86.7	263.8	106.84	20.40
W4 / Dynamic activation precision																
DyQ-VLA	**98.8**	98.4	**98.0**	**96.0**	**97.8**	**149.5**	**55.20**	**13.23**	89.4	**93.4**	**90.4**	**80.8**	**88.5**	**219.8**	**85.07**	**18.04**

*Note: Translation/rotation errors are in mm/degrees. **Bold**/underline denote the best/second-best results.

Kernel Flow Fusion and System-Level Scheduling Optimization. To reduce intermediate data movement, we fuse dequantization and compatible RoPE, GELU, gating, and residual operations with GEMM, using epilogue fusion where applicable. Intermediate results stay in registers or shared memory where possible, reducing HBM writes and reads. Fixed-shape kernels and preallocated buffers support CUDA Graph replay of the selected precision path, reducing repeated scheduling and host kernel launch overhead. Details and illustrations of overall runtime optimization are presented in Appendix D. Together, these optimizations enable low-overhead precision switching and translate dynamic quantization into practical GPU acceleration.

5 Experiments

5.1 Experimental Setup

We evaluate *DyQ-VLA* on $\pi_{0.5}$ (Physical Intelligence et al., 2025) and GR00T-N1.5 (Bjorck et al., 2025) on LIBERO (Liu et al., 2023), with additional $\pi_{0.5}$ tests on RoboTwin 2.0 (Chen et al., 2025) and an Agilex Piper-X arm. We compare against BF16, five general PTQ methods (Frantar et al., 2022; Lin et al., 2024b; Shao et al., 2024; Xiao et al., 2023; Lin et al., 2024a), and four VLA quantization baselines (Akbari et al., 2026; Xu et al., 2026c; Zhang et al., 2026a; Wang et al., 2026c) (Tab. 1). All experiments run on an NVIDIA A100 GPU; *DyQ-VLA* uses W4 weights with dynamic A2/A4/A8/A16 activations. SR covers all rollouts; execution steps and end-effector errors relative to matched demonstrations are averaged over successful episodes. See more details in Appendix B.

5.2 Evaluating Results

Simulation Results. On LIBERO (Tab. 1), *DyQ-VLA* achieves the highest average SR among quantized methods: 97.8% on $\pi_{0.5}$ and 88.5% on GR00T-N1.5, exceeding the corresponding BF16 results by 0.8 and 2.4 percentage points, respectively. Compared with QuantVLA W4A8, the strongest baseline by average SR (97.7% and 87.7%, respectively), *DyQ-VLA* maintains comparable performance on $\pi_{0.5}$, while its improvement on GR00T-N1.5 comes primarily from Long, where SR increases from 73.4% to 80.8%. On RoboTwin 2.0 (Tab. 2), $\pi_{0.5}$ with *DyQ-VLA* achieves 82.9% and 76.5% SR under Clean and Randomized settings, respectively. These results surpass the strongest quantized baseline in each setting—QuantVLA W4A8 and HoloQ-VLA—by 1.5 and 1.3 percentage points, respectively. Relative to BF16, SR is 0.2 percentage points higher under Clean and 0.3

Table 2: **Task Performance and Inference Efficiency on RoboTwin 2.0 and LIBERO.**

Method	Prec.	RoboTwin 2.0 / $\pi_{0.5}$									LIBERO					
		Clean			Randomized			Steps↓	Speedup↑		$\pi_{0.5}$			GR00T-N1.5		
		SR↑	Trans.	Rot.	SR↑	Trans.	Rot.		Step	Rollout	Step	Rollout	Mem.↓	Step	Rollout	Mem.↓
BF16 ref.	BF16	82.7	32.53	8.77	76.8	39.82	12.27	155.7	1.00	1.00	1.00	1.00	6.71	1.00	1.00	5.45
ActQuant	W4A16	80.9	41.65	12.10	74.2	52.38	16.48	186.4	1.11	0.93	1.10	0.92	**2.63**	1.09	0.94	4.03
QVLA	W4A16	80.5	42.18	10.92	73.7	51.74	15.28	188.2	1.09	0.90	1.08	0.89	2.70	1.07	0.93	4.10
QuantVLA	W4A8	81.4	40.36	15.24	74.9	50.81	20.31	181.7	1.30	1.11	1.32	1.13	3.50	1.23	1.09	4.27
HoloQ-VLA	W4A4	81.0	46.73	13.57	75.2	58.92	18.61	184.8	1.16	0.98	1.17	0.99	3.54	1.10	0.99	**3.99**
DyQ-VLA	W4/A_{dyn}	**82.9**	**34.08**	**9.06**	**76.5**	**42.37**	**12.68**	**147.4**	**1.80**	**1.90**	**1.81**	**1.93**	3.49	**1.74**	**1.88**	4.26

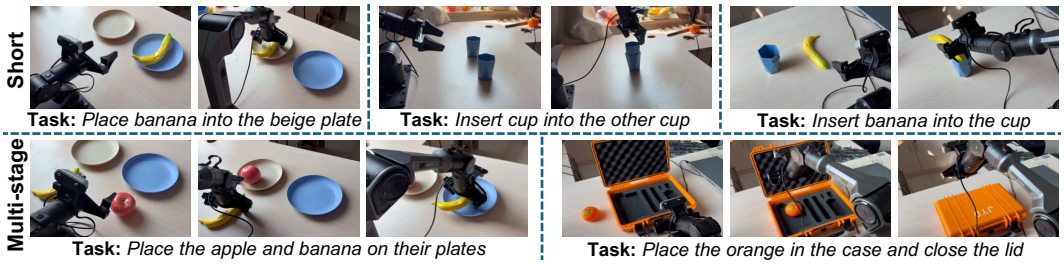

Figure 5: **Real-World Setup and Task Examples.** Details can be found in Appendix F.

percentage points lower under Randomized. These results support the ability of *DyQ-VLA* to maintain task performance across different backbones and simulation environments.

DyQ-VLA reduces rollout inference cost through fewer execution steps. Tab. 2 compares complete inference systems, including quantization methods and operator implementations, with rollout speedup measured from cumulative model inference time. Lower per-step latency supports timely policy updates, but quantization-induced deviations may require additional corrective steps even when policy robustness and task tolerance preserve success. For $\pi_{0.5}$ on LIBERO, ActQuant and QuantVLA achieve per-step speedups of 1.10× and 1.32×, yet longer executions reduce their rollout speedups to 0.92× and 1.13×, respectively.

Compared with QuantVLA W4A8, *DyQ-VLA* reduces mean execution steps per successful episode by 19.8%, 17.7%, and 18.9% for $\pi_{0.5}$ and GR00T-N1.5 on LIBERO and $\pi_{0.5}$ on RoboTwin 2.0, respectively. Steps also decrease by 6.4%, 7.3%, and 5.3% relative to BF16. BF16 itself deviates from the reference demonstrations, so full precision does not guarantee the fewest execution steps. Combining fewer steps with faster per-step inference, *DyQ-VLA* achieves measured rollout speedups of 1.88×-1.93× over BF16, exceeding its per-step speedups of 1.74×-1.81×. Rollout inference is 70.8%-72.5% faster than QuantVLA W4A8, the fastest baseline, at comparable static full-model weight footprints on LIBERO: 3.49 versus 3.50 GB for $\pi_{0.5}$ and 4.26 versus 4.27 GB for GR00T-N1.5.

Table 3: Real-world performance of *DyQ-VLA*, quantization baselines, and the BF16 reference.

Method	Prec.	Short-horizon				Multi-stage		SR (%)↑	Task Spd.↑
		Banana Plate	Cup Insertion	Banana Cup	Apple Plate	Orange Packing	Ordered Placement		
BF16 ref.	BF16	9/10	6/10	8/10	8/10	7/10	7/10	75.0	1.00
QuantVLA	W4A8	8/10	6/10	8/10	8/10	6/10	5/10	68.3	1.16
ActQuant	W4A16	8/10	6/10	8/10	8/10	6/10	5/10	68.3	1.08
QVLA	W4A16	8/10	4/10	8/10	8/10	5/10	5/10	63.3	1.04
HoloQ-VLA	W4A4	8/10	6/10	8/10	8/10	7/10	6/10	71.7	1.11
DyQ-VLA	W4/A_{dyn}	8/10	7/10	8/10	7/10	7/10	7/10	**73.3**	**1.59**

Real-World Evaluation. Across six real-world tasks spanning short-horizon and multi-stage manipulation (Tab. 3), *DyQ-VLA* achieves the highest average SR among quantized methods at 73.3%, close to BF16's 75.0%. On each of the two multi-stage tasks, *DyQ-VLA* succeeds in 7 out of 10 trials, matching BF16. While maintaining this task performance, *DyQ-VLA* achieves a 1.59× end-to-end task completion speedup over BF16, compared with 1.16× for QuantVLA, the fastest quantized baseline, demonstrating its efficiency advantage in real-world execution.

5.3 Discussion

Runtime Error Evaluation. Tab. 4 shows that *DyQ-VLA* achieves the lowest translation and rotation errors and requires the fewest steps on average across all four LIBERO suites, with pronounced gains on long-horizon tasks. On LIBERO-Long, *DyQ-VLA* and QuantVLA W4A8 both achieve 96.0% SR, but *DyQ-VLA* reduces translation and rotation errors relative to demonstrations by 32.1% and 47.2%, respectively, and execution steps from 224.5 to 163.3, a 27.3% reduction. Fig. 6 plots SR against the execution step budget. Under a limited budget, *DyQ-VLA* achieves higher SR, indicating that more tasks are completed within the same number of steps. The curve complements the final SR comparison by showing differences in execution length, consistent with the lower step counts reported in the table.

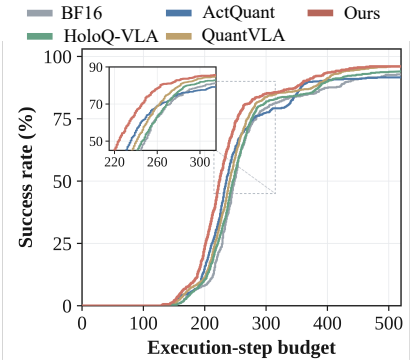

Figure 6: SR versus steps.

Table 4: Runtime evaluation on LIBERO ($\pi_{0.5}$).

Method	Spatial				Object			
	SR↑	Trans.↓	Rot.↓	Steps↓	SR↑	Trans.↓	Rot.↓	Steps↓
ActQuant	**98.8**	63.82	12.42	177.0	99.0	64.95	14.56	174.0
QuantVLA	98.7	61.44	13.74	174.0	98.0	66.18	15.08	171.0
HoloQ-VLA	97.0	76.80	13.18	176.0	**100.0**	82.20	14.21	174.0
DyQ-VLA	**98.8**	**53.58**	**11.45**	**145.0**	98.4	**55.85**	**13.18**	**143.0**
	Goal				**Long**			
ActQuant	96.4	65.70	15.18	181.0	91.6	93.85	19.92	235.8
QuantVLA	**98.0**	61.72	17.26	177.0	**96.0**	91.98	26.60	224.5
HoloQ-VLA	96.0	80.40	15.62	179.0	94.0	99.48	23.55	231.1
DyQ-VLA	**98.0**	**48.91**	**14.25**	**147.0**	**96.0**	**62.45**	**14.04**	**163.3**

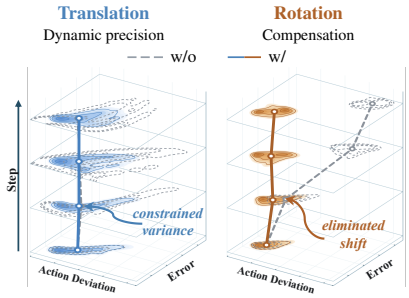

Figure 7: Translational and Rotational Error Distribution Ablation.

Ablation Study. Compensation (Comp.) raises W4A4 SR from 94.1% to 96.3% (Tab. 5). With dynamic precision, it raises SR from 97.2% to 97.8%, reduces mean steps from 155.1 to 149.5, and improves rollout speedup from 1.89× to 1.93×. Shuffling precision order (A_{shuf}) with compensation increases errors and steps, reducing SR to 96.7% and rollout speedup to 1.82×. Fig. 7 shows that dynamic precision constrains translation error variance, while compensation suppresses shifts in rotation error distributions across steps.

Table 5: Ablation on $\pi_{0.5}$ across LIBERO suites. Rollout denotes measured speedup over BF16.

Prec.	Comp.	SR ↑	Error ↓		Steps ↓	Rollout ↑
			Trans.	Rot.		
BF16 ref.	—	97.0	60.37	14.19	159.7	1.00
W4A8	×	94.4	74.94	19.06	168.2	1.29
W4A4	×	94.1	91.47	20.75	171.5	1.44
W4A4	✓	96.3	68.72	15.81	160.8	1.49
W4/A_{dyn}	×	97.2	61.84	14.47	155.1	1.89
W4/A_{shuf}	✓	96.7	65.84	15.26	158.4	1.82
W4/A_{dyn}	✓	**97.8**	**55.20**	**13.23**	**149.5**	**1.93**

6 Conclusion

This paper analyzes the correlation between actions and errors in closed-loop VLA execution, revealing that quantization disperses the translation error distribution and induces cumulative drift in rotation errors. Based on this observation, we propose *DyQ-VLA*, which dynamically selects activation precision and compensates accumulated residuals to simultaneously control quantization overhead and runtime errors. We believe *DyQ-VLA* facilitates future VLA deployment.

AI USE STATEMENT

We use AI solely for language polishing and improving the clarity of the manuscript. All technical content, experimental design, analysis, and conclusions were developed and verified by the authors.

REFERENCES

Arash Akbari, Arman Akbari, Masih Eskandar, Qitao Tan, Yixiao Chen, Jingwu Luo, Bertha Pangaribuan, Liyun Zhang, Jennifer Dy, Geng Yuan, et al. Actquant: Sub-4-bit action-guided quantization for vision-language-action models. *arXiv preprint arXiv:2605.24011*, 2026.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

Muhayy Ud Din, Waseem Akram, Lyes Saad Saoud, Jan Rosell, and Irfan Hussain. Vision language action models in robotic manipulation: A systematic review. *arXiv preprint arXiv:2507.10672*, 2025.

Hengyu Fang, Yijiang Liu, Yuan Du, Li Du, and Huanrui Yang. Sqap-vla: A synergistic quantization-aware pruning framework for high-performance vision-language-action models. *arXiv preprint arXiv:2509.09090*, 2025.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Titong Jiang, Xuefeng Jiang, Yuan Ma, Xin Wen, Bailin Li, Kun Zhan, Peng Jia, Yahui Liu, Sheng Sun, and Xianpeng Lang. The better you learn, the smarter you prune: Towards efficient vision-language-action models via differentiable token pruning. *arXiv preprint arXiv:2509.12594*, 2025.

Kinam Kim, Namiko Saito, Heecheol Kim, Katsushi Ikeuchi, Jaegul Choo, and Yasuyuki Matsushita. Object-centric residual rl for zero-shot sim-to-real vla enhancement. *arXiv preprint arXiv:2606.18953*, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024. URL https://simpler-env.github.io/.

Ziwen Li, Xin Wang, Hanlue Zhang, Runnan Chen, Runqi Lin, Xiao He, Han Huang, Yandong Guo, Fakhri Karray, Tongliang Liu, et al. Posa-vla: Enhancing action generation via pose-conditioned anchor attention. *arXiv preprint arXiv:2512.03724*, 2025.

Haokun Lin, Haobo Xu, Yichen Wu, Jingzhi Cui, Yingtao Zhang, Linzhan Mou, Linqi Song, Zhenan Sun, and Ying Wei. Duquant: Distributing outliers via dual transformation makes stronger quantized llms. *Advances in Neural Information Processing Systems*, 37:87766–87800, 2024a.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6: 87–100, 2024b.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Weijia Liufu, Xiaoyu Guo, Ruiyi Chen, Jingzhi Liu, Kaidong Zhang, Xiwen Liang, Jianqi Lin, Dawei Sun, Yuze Wang, Rongtao Xu, et al. Repo-vla: Recovery-driven policy optimization for vision-language-action models. *arXiv preprint arXiv:2605.09410*, 2026.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. RoboCasa: Large-scale simulation of everyday tasks for generalist robots. In *Robotics: Science and Systems (RSS)*, 2024. URL https://robocasa.ai/.

Open X-Embodiment Collaboration. Open X-embodiment: Robotic learning datasets and RT-X models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903, 2024. URL https://robotics-transformer-x.github.io/.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a Vision-Language-Action model with Open-World generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Navin Ranjan and Andreas Savakis. Mix-qvla: Task-evidence-aware mixed-precision quantization of vision-language-action models. *arXiv preprint arXiv:2606.19565*, 2026.

Robert Ramirez Sanchez, Daniel J Evans, Dylan P Losey, and Siddarth Jain. Volt: Vision and language trajectory segmentation for faster-than-demonstration policies. *arXiv preprint arXiv:2606.06323*, 2026.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=8Wuvhh0LYW.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

Guodong Wang, Chenkai Zhang, Qingjie Liu, Jinjin Zhang, Jiancheng Cai, Junjie Liu, and Xinmin Liu. Libero-x: Robustness litmus for vision-language-action models. *arXiv preprint arXiv:2602.06556*, 2026a.

Hanzhen Wang, Jiaming Xu, Yushun Xiang, Jiayi Pan, Yongkang Zhou, Yong-Lu Li, and Guohao Dai. Specprune-vla: Accelerating vision-language-action models via action-aware self-speculative pruning. *arXiv preprint arXiv:2509.05614*, 2025a.

Hongyu Wang, Chuyan Xiong, Ruiping Wang, and Xilin Chen. Bitvla: 1-bit vision-language-action models for robotics manipulation. *arXiv preprint arXiv:2506.07530*, 2025b.

Songsheng Wang, Rucheng Yu, Zhihang Yuan, Chao Yu, Feng Gao, Yu Wang, and Derek F Wong. Spec-vla: speculative decoding for vision-language-action models with relaxed acceptance. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 26916–26928, 2025c.

Xianghui Wang, Feng Chen, Wenbo Zhang, Hua Yan, Zixuan Wang, Changsheng Li, and Yinjie Lei. Policytrim: Boosting intrinsic policy efficiency of vision-language-action models. In *European Conference on Computer Vision*, pp. 93–110. Springer, 2026b.

Xinyu Wang, Mingze Li, Sicheng Lyu, Dongxiu Liu, Kaicheng Yang, Ziyu Zhao, Yufei Cui, Xiao-Wen Chang, and Peng Lu. Holoq-vla: Uniform w4a4 quantization of vision-language-action models, 2026c. URL https://arxiv.org/abs/2605.28803.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, et al. Tinyvla: toward fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 10(4):3988–3995, 2025.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pp. 38087–38099. PMLR, 2023.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. Vla-cache: Efficient vision-language-action manipulation via adaptive token caching. *Advances in Neural Information Processing Systems*, 38:164448–164473, 2026a.

Siyuan Xu, Tianshi Wang, Fengling Li, Lei Zhu, and Heng Tao Shen. Da-ptq: Drift-aware post-training quantization for efficient vision-language-action models. *arXiv preprint arXiv:2604.11572*, 2026b.

Yuhao Xu, Yantai Yang, Zhenyang Fan, Yufan Liu, Yuming Li, Bing Li, and Zhipeng Zhang. Qvla: Not all channels are equal in vision-language-action model's quantization. *arXiv preprint arXiv:2602.03782*, 2026c.

Yantai Yang, Yuhao Wang, Zichen Wen, Luo Zhongwei, Chang Zou, Zhipeng Zhang, Chuan Wen, and Linfeng Zhang. Efficientvla: Training-free acceleration and compression for vision-language-action models. *Advances in Neural Information Processing Systems*, 38:40891–40914, 2026.

Yang Yue, Yulin Wang, Bingyi Kang, Yizeng Han, Shenzhi Wang, Shiji Song, Jiashi Feng, and Gao Huang. Deer-vla: Dynamic inference of multimodal large language models for efficient robot execution. *Advances in Neural Information Processing Systems*, 37:56619–56643, 2024.

Dapeng Zhang, Jing Sun, Chenghui Hu, Xiaoyan Wu, Zhenlong Yuan, Rui Zhou, Fei Shen, and Qingguo Zhou. Pure vision language action (vla) models: A comprehensive survey. *arXiv preprint arXiv:2509.19012*, 2025.

Jingxuan Zhang, Yunta Hsieh, Zhongwei Wan, Haokun Lin, Xin Wang, Ziqi Wang, Yingtie Lei, and Mi Zhang. Quantvla: Scale-calibrated post-training quantization for vision-language-action models. *arXiv preprint arXiv:2602.20309*, 2026a.

Rongyu Zhang, Menghang Dong, Yuan Zhang, Liang Heng, Xiaowei Chi, Gaole Dai, Li Du, Dan Wang, Yuan Du, and Shanghang Zhang. Mole-vla: Dynamic layer-skipping vision language action model via mixture-of-layers for efficient robot manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 18764–18772, 2026b.

Ganlong Zhao, Zijia Tang, Xingping Chen, Zhanghui Kuang, Ye Tian, and Guanbin Li. Flare: A failure-aware framework for autonomous correction and recovery in visual-language robotic manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22391–22401, 2026.

Zihao Zheng, Zhihao Mao, Maoliang Li, Jiayu Chen, Xinhao Sun, Zhaobo Zhang, Donggang Cao, Hong Mei, and Xiang Chen. Kerv: Kinematic-rectified speculative decoding for embodied vla models. *arXiv preprint arXiv:2603.01581*, 2026.

A Extended Related Work

A.1 Efficient Inference for VLA Models.

To reduce the computational and memory costs of VLA inference, existing studies explore three broad directions: (1) model compression, (2) runtime optimization, and (3) architectural design.

Model Compression. Compression methods reduce redundant computation in action prediction through visual token selection and model layer pruning (Yang et al., 2026; Wang et al., 2025a; Jiang et al., 2025). For example, LightVLA learns token selection through differentiable pruning, retaining visual information useful for task execution (Jiang et al., 2025).

Runtime optimizations. Runtime optimizations improve how computation is executed and reused. Speculative decoding reduces sequential decoding costs by rapidly drafting action tokens and verifying them in parallel (Wang et al., 2025c; Zheng et al., 2026). Caching methods reuse visual computation across observations or intermediate features across diffusion denoising steps (Xu et al., 2026a; Yang et al., 2026). Early exiting and dynamic layer skipping further adapt the executed model depth to computational budgets or inputs (Yue et al., 2024; Zhang et al., 2026b).

Architecture Design. Architectural approches reduce model costs through compact multimodal backbones and dedicated action modules, as exemplified by TinyVLA (Wen et al., 2025) and SmolVLA (Shukor et al., 2025). Action representations and prediction mechanisms are also redesigned: FAST compresses action token sequences (Pertsch et al., 2025), while OpenVLA-OFT combines action chunking with parallel decoding to accelerate action generation (Kim et al., 2025).

A.2 Quantization of Vision-Language-Action Models

To improve the accuracy-speed trade-off of low-precision VLA inference, existing studies explore three directions: (1) distribution-aware quantization, (2) sensitivity-aware quantization, and (3) trajectory-aware quantization.

Distribution-Aware Quantization. Distribution-aware methods mitigate quantization distortion through representation adaptation and numerical calibration. BitVLA (Wang et al., 2025b) combines quantization with distillation to preserve model capabilities, while QuantVLA (Zhang et al., 2026a) calibrates attention temperatures and residual output scales. HoloQ-VLA (Wang et al., 2026c) addresses weight and activation distortion through composite rotations and denoising-step-dependent scaling. SQAP-VLA (Fang et al., 2025) further coordinates quantization with visual token pruning, accounting for the effects of quantized representations on token selection.

Sensitivity-Aware Quantization. Sensitivity-aware methods assign different bit widths to model components according to their influence on action prediction. QVLA (Xu et al., 2026c) uses action output sensitivity to guide channel-wise bit allocation and validates this sensitivity measure over short horizons. ActQuant (Akbari et al., 2026) similarly allocates precision according to action relevance, assigning a bit width to each weight matrix based on its contribution to action prediction.

Trajectory-Aware Quantization. Trajectory-aware methods further incorporate trajectory drift and execution-stage information into offline precision allocation. DA-PTQ (Xu et al., 2026b) combines representation compensation with drift-aware layerwise bit allocation, using offline estimates of trajectory drift to identify components that require higher precision. Mix-QVLA (Ranjan & Savakis, 2026) uses task evidence and phase-dependent sensitivity to guide layerwise precision selection, accounting for changes in layer importance across task execution. Both methods use trajectory or temporal information during calibration to determine precision configurations that remain fixed during deployment.

A.3 Core Differences Between *DyQ-VLA* and Prior Works

SOTA Quantization Works. Prior studies follow conventional approaches to large-model quantization, focusing primarily on precision selection and the accuracy-speed trade-off. Although some methods consider the closed-loop nature of VLA execution, how runtime errors evolve and how quantization alters the action-error correlation remain insufficiently characterized.

***DyQ-VLA*'s Key Innovation.** In contrast, we explicitly analyze the action-error correlation, reveal its distinct characteristics in the translational and rotational components, and characterize the impact of quantization on this correlation. Building on these findings, we rethink quantization strategy design for VLA models and propose *DyQ-VLA*, a runtime-error-aware dynamic quantization framework.

B Definitions and Measurement of Runtime Error

B.1 Sources and Measurement of Runtime Error

Runtime error can arise from imperfect action prediction. VLA models learn a mapping from observations to actions using finite demonstration data. Limited data coverage and imperfect model fitting can cause predicted translation and rotation actions to differ from reference actions during deployment. Executing these actions can therefore produce poses that deviate from the reference trajectory. Finite controller tracking accuracy and contact constraints can also cause actual motion to differ from commanded motion. These deviations alter subsequent observations, potentially leading to states poorly covered by demonstrations and affecting later action predictions. Quantization approximates weights and activations, further altering action outputs and the evolution of execution deviations. We measure runtime error through the discrepancy between the actual and reference poses of the end effector. We measure this deviation using end-effector poses after each execution chunk. Let t index policy queries, with the first h predicted actions executed before the next query. Denote the resulting position and unit quaternion by $\mathbf{p}_t$ and $\mathbf{q}_t$, and the corresponding reference pose in the same coordinate frame by $\bar{\mathbf{p}}_t$ and $\bar{\mathbf{q}}_t$. Translation and rotation errors are defined as

$$\begin{aligned} e_t^T &= \|\mathbf{p}_t - \bar{\mathbf{p}}_t\|_2, \\ e_t^R &= 2\arccos\big(\min\{1, |\mathbf{q}_t^\top \bar{\mathbf{q}}_t|\}\big). \end{aligned} \tag{5}$$

The rotation error is the shortest angle between the two orientations. The absolute value accounts for quaternion sign ambiguity, while clipping prevents numerical round-off from producing invalid inputs to $\arccos$. Translation and rotation errors are computed in meters and radians and reported in millimeters and degrees, respectively.

On LIBERO, $h = 5$ for $\pi_{0.5}$ and $h = 1$ for GR00T-N1.5. The query index t is distinct from control-step and internal flow-step indices. For action–error analysis, the error measured after executing the actions from query t is paired with the action predictions at query $t + 1$.

B.2 Reference Trajectories

Task evaluation uses matched and aligned demonstration trajectories as references for both BF16 and quantized models. Success rate is computed over all episodes, whereas runtime errors and execution steps are averaged over successful episodes. In the LIBERO main table, errors are averaged equally across the four suite-level means, while execution steps are aggregated using the number of successful episodes in each suite as weights. Action–Error Correlation and sensitivity analyses use unperturbed rollouts at the same precision as references. The reference and perturbed branches share the same initial state and matched policy randomness, and each proceeds in a closed loop using its own observations after the perturbation. We pair pose deviations at corresponding execution steps with action-prediction differences at the next policy query, and characterize the effects of quantization through changes in the joint distributions and sensitivities across precision settings.

B.3 Action-Error Joint Distributions

Following the paired rollout protocol in Appendix B.2, we inject action perturbations of equal magnitude at a fixed policy query and record subsequent execution errors and differences in action predictions. The normalized error after executing query t and action discrepancy at the next query are

$$\begin{aligned} \varepsilon_t &= \sqrt{\left(\frac{e_t^T}{s_T}\right)^2 + \left(\frac{e_t^R}{s_R}\right)^2}, \\ \Delta A_{t+1} &= \sqrt{\frac{1}{h}\sum_{i=1}^{h}\sum_{d=1}^{6}\left(\frac{a_{t+1,i,d} - \bar{a}_{t+1,i,d}}{s_d^A}\right)^2}. \end{aligned} \tag{6}$$

Here, a and $\bar{a}$ denote commands for the end effector passed to the LIBERO controller by the perturbed and reference branches, respectively, both operating at the same precision. The six channels comprise three translation and three rotation components, excluding the gripper. The scales s_T, s_R, and s_d^A are calibrated as described in Appendix B.6 and held fixed across comparisons. Each paired rollout contributes a sample $(\Delta A_{t+1}, \varepsilon_t)$ at query t. For temporal comparisons, we retain a fixed set of N paired rollouts with valid records at every displayed checkpoint.

At a fixed query, let $x_n = \Delta A_{t+1}^{(n)}$ and $y_n = \varepsilon_t^{(n)}$. We estimate their joint density using Gaussian kernels with reflection at the zero boundaries:

$$\hat{p}_t(x,y) = \frac{1}{N}\sum_{n=1}^{N}\big[K_x(x-x_n)+K_x(x+x_n)\big]\big[K_y(y-y_n)+K_y(y+y_n)\big], \qquad x,y \geq 0, \quad (7)$$

where

$$K_x(u) = \frac{1}{\sqrt{2\pi}\sigma_x}\exp\left(-\frac{u^2}{2\sigma_x^2}\right),$$

and K_y is defined analogously. Each bandwidth is the corresponding coordinate's standard deviation pooled across checkpoints, multiplied by $N^{-1/6}$, with a lower bound equal to the larger of 2% of its pooled maximum and 10^{-4}. Bandwidths remain fixed across checkpoints for each component. Distribution centers and dispersion are measured by the means and variances of the original samples.

We further extract translation and rotation components from the same paired rollouts. Let $D_T = \{1,2,3\}$ and $D_R = \{4,5,6\}$ denote the corresponding action channels. For $X \in \{T, R\}$, we define

$$\varepsilon_t^X = \frac{e_t^X}{s_X}, \qquad \Delta A_{t+1}^X = \sqrt{\frac{1}{h}\sum_{i=1}^{h}\sum_{d\in D_X}\left(\frac{a_{t+1,i,d}-\bar{a}_{t+1,i,d}}{s_d^A}\right)^2}. \quad (8)$$

Applying the same density estimator to $(\Delta A_{t+1}^T, \varepsilon_t^T)$ and $(\Delta A_{t+1}^R, \varepsilon_t^R)$ yields the translation and rotation action-error distributions, respectively, for comparing their centers and dispersion across execution.

B.4 Signed Rotational Residuals

Let $\mathbf{x}_{k,n,i}$ denote the monitored activation for action i at flow step n of query k. The signed quantization residual is

$$\boldsymbol{\delta}_{k,n,i} = \mathrm{QDQ}(\mathbf{x}_{k,n,i}) - \mathbf{x}_{k,n,i}, \quad (9)$$

where QDQ denotes quantization followed by dequantization at the current precision. We project these residuals onto rotational components and accumulate them across queries:

$$\boldsymbol{\zeta}_k^R = \overline{\mathbf{B}}_R \sum_{n=1}^{N_f}\Delta\tau_n \sum_{i=1}^{h}\boldsymbol{\delta}_{k,n,i}, \qquad \boldsymbol{\mathcal{R}}_t^R = \sum_{k=0}^{t-1}\boldsymbol{\zeta}_k^R. \quad (10)$$

Here, N_f is the number of flow steps and $\Delta\tau_n$ is the sampler's signed integration coefficient. The fixed matrix $\overline{\mathbf{B}}_R$ averages local Jacobians from monitored activations to rotational flow outputs over calibration samples and flow steps. Accumulation uses a common coordinate frame and fixed scales, preserving reinforcement and cancellation between residuals. For error tracking, we pair e_t^R, measured after executing query t's actions, with $\|\boldsymbol{\mathcal{R}}_{t+1}^R\|_2$, which includes the current query's contribution.

B.5 Response Sensitivity

At each precision, response sensitivity measures the change in subsequent action predictions per unit normalized execution error. For paired samples with perturbation direction j,

$$\mathscr{S}_{t,j}^X = \frac{\Delta A_{t+1}}{\varepsilon_t^X}, \qquad X \in \{T,R\}, \quad \varepsilon_t^X > 0. \quad (11)$$

Here, ΔA_{t+1} and ε_t^X follow Appendix B.3. The numerator includes both translation and rotation action changes to capture responses across components, while the denominator uses the corresponding normalized error. We compare FP and quantized sensitivities by query and perturbation direction under matched tasks, initial states, and perturbation conditions, using identical normalization scales to characterize quantization effects on action-error responses.

B.6 Calibration of Scales

We calibrate scales using 512 FP LIBERO rollouts per backbone, initialized from training demonstrations without injected perturbations, approximately balanced across analyzed tasks, and separate from evaluation. For each execution chunk c in the calibration set $\mathcal{C}$, let Δp_c and $\Delta\theta_c$ denote the translation and rotation magnitudes between its initial and final poses, computed using Eq. (5). With Q_α denoting the empirical α quantile, we define:

$$\begin{aligned} s_T &= Q_{0.95}(\{\Delta p_c : c \in \mathcal{C}\}), \\ s_R &= Q_{0.95}(\{\Delta\theta_c : c \in \mathcal{C}\}), \\ s_d^A &= \frac{Q_{0.99}(a_d) - Q_{0.01}(a_d)}{2}. \end{aligned} \tag{12}$$

Here, a_d contains the executed values of action channel d, whose scale follows percentile normalization (Kim et al., 2024; Pertsch et al., 2025). For each backbone, scales remain fixed across precision settings, queries, and perturbation conditions. Normalized differences are not clipped.

C Additional Runtime Error Analysis

C.1 Trajectory Visualizations.

We provide trajectory visualizations to complement the runtime error analysis in Sec. 3. Fig. 8 (a) compares the FP reference, open-loop evaluation, and closed-loop execution trajectories, illustrating how action errors evolve through feedback between the policy and its environment. In Fig. 8 (b), the same action offset is injected at different execution steps, producing distinct trajectories: one rollout recovers from the resulting error, while the other fails. Fig. 8 (c) fixes the injection step and offset magnitude while varying the direction, yielding different trajectories and task outcomes. Even offsets of the same magnitude affect subsequent behavior differently depending on the execution state and offset direction.

C.2 Action-Error Distributions Across Models and Benchmarks

We extend the analysis in Sec. 3.1 to GR00T-N1.5 on the four LIBERO suites and $\pi_{0.5}$ on RoboTwin 2.0 under Clean and Randomized settings, examining whether the component differences persist across models and environments under full precision. Following the paired rollout and density estimation procedures in Sec. B.3, we construct joint distributions of runtime error and subsequent action discrepancy for translation and rotation, and compare their evolution across policy queries.

Fig. 9 shows a consistent pattern across all six settings. Translation distributions remain relatively dispersed around stable centers, whereas rotation distributions are more concentrated and their centers progressively shift toward larger errors. These results support Observation ① in Sec. 3.1 across model backbones, task types, and benchmark settings.

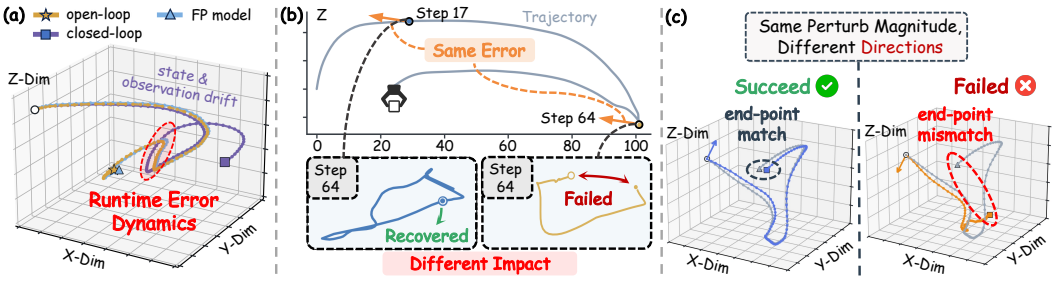

Figure 8: **Trajectory Visualizations of Runtime Error Dynamics and Their Behavioral Impact.** (a) FP reference, open-loop evaluation, and closed-loop execution trajectories. (b) Identical action offsets at different steps, with trajectories projected onto the yz plane. (c) Offsets of equal magnitude in different directions at the same step.

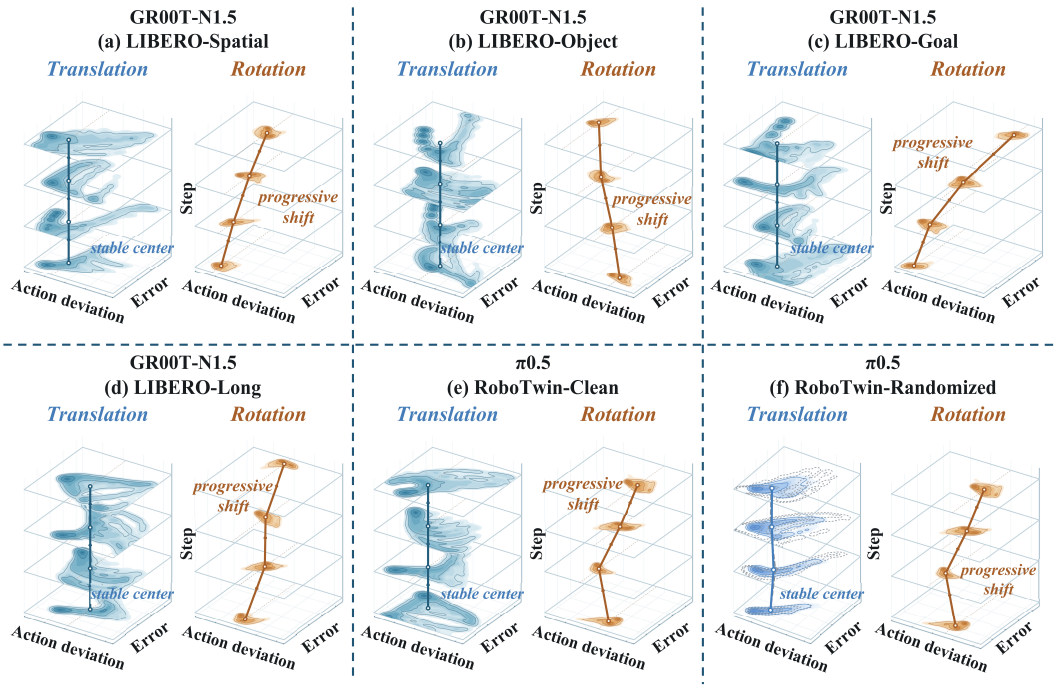

Figure 9: **Action-Error Distributions Across Models and Benchmarks.**

C.3 Tracking Runtime Error with Quantization Residuals

We evaluate whether quantization residuals track rotational runtime error using 200 trajectories across four LIBERO suites, with 10 tasks per suite and five initial states per task. We compare current residual magnitude, unsigned accumulation, and signed accumulation, aligned with the rotational error e_t^R measured after executing query t. For each trajectory, we compute Spearman correlations with error levels and with error changes between successive queries. Fig. 10(a) shows that unsigned and signed accumulation both correlate strongly with error levels. However, unsigned accumulation tracks error changes less consistently, with median change correlations near zero on Object and Long, whereas signed accumulation maintains high correlations across all suites (Fig. 10(b)).

This difference follows from how residual history is aggregated. Current magnitude ignores historical contributions, while unsigned accumulation discards directional information. Let $u_t = \sum_{k=0}^{t} \|\boldsymbol{\zeta}_k^R\|_2$ denote unsigned accumulation and $s_t = \|\boldsymbol{\mathcal{R}}_{t+1}^R\|_2$ denote the magnitude of signed accumulation,

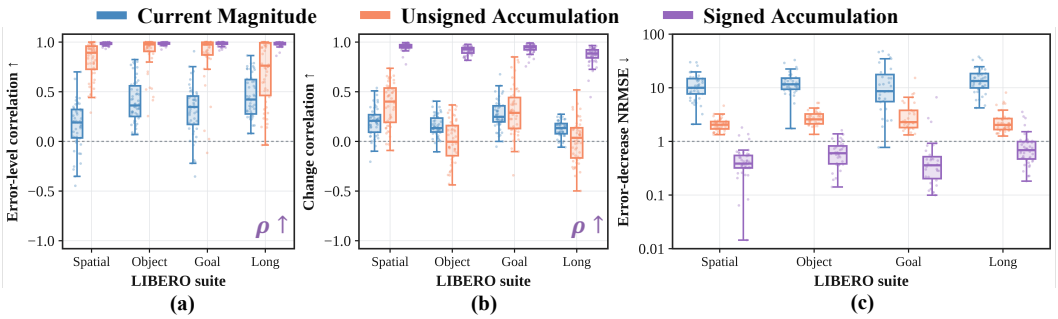

Figure 10: **Tracking Rotational Runtime Error.** (a) Spearman correlation with rotational runtime error. (b) Spearman correlation between signal changes and error changes. (c) NRMSE of normalized changes for error decreases $\geq 0.5°$; the dashed line marks the zero-change baseline. Dots denote trajectories; boxes show medians and interquartile ranges, with 1.5 IQR whiskers.

both including query t. The residual definition in Eq. (10) gives

$$\begin{aligned} u_t - u_{t-1} &= \|\boldsymbol{\zeta}_t^R\|_2 \geq 0, \\ s_t^2 - s_{t-1}^2 &= \|\boldsymbol{\zeta}_t^R\|_2^2 + 2\left\langle \boldsymbol{\mathcal{R}}_t^R, \boldsymbol{\zeta}_t^R \right\rangle. \end{aligned} \tag{13}$$

Unsigned accumulation cannot decrease, even when it correlates strongly with overall error levels. Signed accumulation retains the interaction between current and historical residuals: aligned contributions reinforce each other, while sufficiently strong opposing contributions reduce the accumulated magnitude. It therefore captures both reinforcement and cancellation across queries.

We further test whether this ability to decrease corresponds to observed error reductions. Each signal and e_t^R is divided by its own trajectory maximum. At checkpoints where the unnormalized rotation error decreases by at least $0.5°$, NRMSE is the RMS difference between normalized signal and error changes, divided by the RMS normalized error change. Here, normalized error changes are negative, whereas changes in unsigned accumulation remain nonnegative. Consequently, $\mathrm{NRMSE}_u \geq 1$, where one is the baseline for predicting zero change. In Fig. 10(c), signed accumulation achieves median NRMSE below this baseline in every suite, showing that it follows decreases in measured rotational runtime error.

C.4 Feedback Estimation of Response Sensitivity

Fig. 11 shows that feedback estimates track the relative strength of translation response sensitivity across both backbones and all four LIBERO suites. Median Spearman correlations range from 0.83 to 0.92 across queries (Temporal) and from 0.77 to 0.87 across perturbation directions (Directional). We evaluate 40 trajectories per backbone and suite, for 320 trajectories in total. Measured sensitivity follows Eq. (11) with $X = T$: the subsequent action response, including translation and rotation components, per unit normalized translation error. Temporal correlations hold the perturbation direction fixed, while Directional correlations hold the query fixed. These results show that the estimates distinguish sensitivity across both execution stages and perturbation directions.

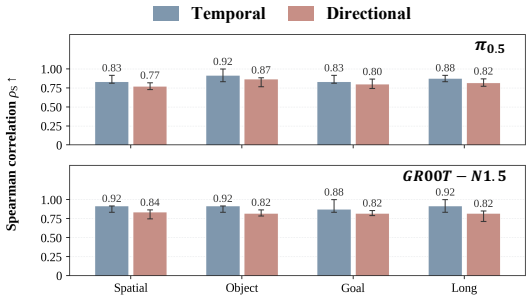

Figure 11: **Feedback Sensitivity Estimation.** Spearman correlations with measured translation response sensitivity across queries (Temporal) and perturbation directions (Directional). Bars show medians across 40 trajectories per backbone and LIBERO suite; error bars span the 25th to 75th percentiles.

The controller derives feedback scores from executed motion and the remaining action plan. Let $\mathbf{a}_{t,i}^T$ and $\mathbf{a}_{t,i}^R$ denote translation and rotation commands at query t. After executing the first h actions,

$$\begin{aligned} \mathbf{c}_t &= \sum_{i=1}^{h} \mathbf{a}_{t,i}^T, \qquad \mathbf{d}_t = \mathbf{p}_t - \mathbf{p}_{t-1}, \\ g_t &= \frac{\mathbf{d}_t^\top \mathbf{c}_t}{\|\mathbf{c}_t\|_2^2 + \epsilon}, \quad G_t = \max\{G_{t-1}, g_t, \epsilon\}. \end{aligned} \tag{14}$$

Here, g_t measures execution gain along the commanded direction, G_t is its running maximum initialized to zero, and $\epsilon = 10^{-12}$. The normalized scores are

$$\begin{aligned} \widehat{s}_{t+1,T} &= \frac{[1 - g_t/(G_t + \epsilon)]_+}{\theta_T}, \\ \widehat{s}_{t+1,R} &= \frac{1}{\theta_R(h-1)} \sum_{i=h+1}^{2h-1} \|\mathbf{a}_{t,i+1}^R - \mathbf{a}_{t,i}^R\|_2. \end{aligned} \tag{15}$$

Here, $[x]_+ = \max(x, 0)$ and $\theta_T, \theta_R > 0$ are fixed normalization thresholds. Lower execution gain and greater rotation variation in the unexecuted tail increase the respective scores, tightening the constraints in Eq. (3) without additional policy evaluations.

For correlation analysis, the feedback estimate $\widehat{\mathscr{S}}^T_{t,j}$ formed after query t is compared with the measured sensitivity $\mathscr{S}^T_{t,j}$ of the subsequent action response:

$$\begin{aligned} \rho_{\text{Temporal}}(j) &= \text{Spearman}_t\left(\widehat{\mathscr{S}}^T_{t,j}, \mathscr{S}^T_{t,j}\right), \\ \rho_{\text{Directional}}(t) &= \text{Spearman}_j\left(\widehat{\mathscr{S}}^T_{t,j}, \mathscr{S}^T_{t,j}\right). \end{aligned} \tag{16}$$

The correlations support feedback-based sensitivity ordering across queries and perturbation directions for dynamic precision selection (Sec. 4.1).

D Method and Implementation Details

D.1 Algorithm Overview

Algorithm 1 summarizes the offline preparation and online execution of *DyQ-VLA*. Offline, we quantize the target weights to 4 bits, calibrate the rotational residual projection, and prepare the execution paths for candidate activation configurations. We profile their error estimates $E_d(b)$ and inference latencies $L(b)$, and calibrate the thresholds τ_d using the reference configuration. These quantities remain fixed during online execution.

Before each policy query, the controller selects a configuration using the sensitivity estimated from previous execution feedback and the residual remaining after compensation. Following Eq. (3), it chooses the lowest-latency configuration that passes the calibrated test, or uses the predefined fallback if none passes. The selected path produces an action chunk and activation residual records $\mathcal{Q}_t$. Their rotational projection is combined with the historical state $\mathcal{H}_{t-1}$ to obtain $\widehat{\mathbf{e}}_t$. The rotation commands are then corrected according to Eq. (4).

After execution, we update the signed residual state using the actual command corrections. Both the residual contribution and the correction are computed over the executed prefix of length $m_t \leq h$, accounting for early termination. The map ρ_R converts the updated state to the scalar residual scale used by the selection criterion, while Ψ estimates the next query's sensitivity from execution feedback. This completes the connection between precision selection, rotation compensation, and subsequent execution.

D.2 Quantized Kernels with Shared Weights

Fig. 12 shows how the A2, A4, A8, and A16 paths share W4 storage on NVIDIA SM80. Quantized weights are packed as two signed INT4 codes per byte, with FP32 scales per output channel. Modules access a single device allocation through views, while each kernel converts its weight tiles into the required operand format on chip.

The A2 path quantizes activations to ternary values and retains two adjacent pairs within each group of eight. The selected values and position metadata form a pairwise 4:8 sparse INT4 operand for `mma.sp`. The A4 path directly uses dense INT4 multiplication. The general A8 path expands W4 fragments to signed INT8 in registers and evaluates each $K = 64$ slice with two INT8 matrix operations. The A16 path reconstructs BF16 weight tiles in shared memory and uses BF16 multiplication with FP32 accumulation. These conversions remain local to the current tile, avoiding additional resident weight copies.

The integer paths accumulate into INT32 registers and apply activation-row and weight-column scales before BF16 output rounding. The A16 path follows its corresponding weight-scaling and rounding convention. Thus, activation precision can change while the resident weights remain in W4 format.

D.3 Kernel Fusion and Runtime Scheduling

As illustrated in Fig. 12, we fuse activation preparation and output processing to reduce intermediate memory traffic. The activation producer combines normalization, affine transformation, row-scale

Algorithm 1 Runtime Error Control with *DyQ-VLA*

Require: Model π, instruction ℓ, calibration data $\mathcal{D}_{\text{cal}}$; candidate configurations $\mathcal{B}$, reference b_{ref}, fallback $b_{\text{fb}} \in \mathcal{B}$; execution horizon h, gain $\eta \in (0, 1]$; calibrated sensitivity estimator Ψ, residual-scale map ρ_R, and initial sensitivity $\widehat{\mathbf{s}}_1 \geq \mathbf{0}$.

Ensure: Action chunks executed with dynamic precision and rotation compensation.

Offline preparation
$\mathbf{W}_4 \leftarrow \textsc{QuantizeWeights}(\pi, \mathcal{D}_{\text{cal}})$
$\overline{\mathbf{B}}_R \leftarrow \textsc{CalibrateProjection}(\pi, \mathcal{D}_{\text{cal}})$
$\{\mathcal{G}_b\}_{b\in\mathcal{B}} \leftarrow \textsc{PrepareRuntime}(\pi, \mathbf{W}_4, \mathcal{B}, \mathcal{D}_{\text{cal}})$
Profile $E_d(b) \geq 0$ and $L(b)$ for $b \in \mathcal{B}$ and $d \in \{T, R\}$.
Calibrate positive thresholds τ_d using b_{ref}.
Online execution
$\mathcal{H}_0 \leftarrow \mathbf{0}$, $r_{0,T} \leftarrow 0$, $r_{0,R} \leftarrow 0$, $t \leftarrow 1$
while the task is not terminated **do**
  $o_t \leftarrow \textsc{Observe}()$
  $\mathcal{B}_t \leftarrow \left\{ b \in \mathcal{B} : \max_{d\in\{T,R\}} \dfrac{\widehat{s}_{t,d}\big(E_d(b) + r_{t-1,d}\big)}{\tau_d} \leq 1 \right\}$
  if $\mathcal{B}_t \neq \varnothing$ **then**
    $b_t \leftarrow \arg\min_{b\in\mathcal{B}_t} L(b)$
  else
    $b_t \leftarrow b_{\text{fb}}$ {No candidate passed the calibrated test.}
  end if
  $(\mathbf{A}_t, \mathcal{Q}_t) \leftarrow \textsc{Replay}(\mathcal{G}_{b_t}, o_t, \ell)$
  $\widehat{\mathbf{e}}_t \leftarrow \mathcal{H}_{t-1} + \textsc{ProjectResidual}(\mathcal{Q}_t, \overline{\mathbf{B}}_R, h)$
  $\widetilde{\mathbf{A}}_{t,1:h} \leftarrow \textsc{CompensateRotation}(\mathbf{A}_{t,1:h}, \widehat{\mathbf{e}}_t, \eta)$ using Eq. (4).
  $(\mathcal{F}_t, \mathbf{A}^{\text{appl}}_{t,1:m_t}, m_t) \leftarrow \textsc{ExecuteAndRecord}(\widetilde{\mathbf{A}}_{t,1:h})$
  $\Delta\mathbf{a}_t^{R,\text{appl}} \leftarrow \sum_{i=1}^{m_t} \left(\mathbf{a}_{t,i}^{R,\text{appl}} - \mathbf{a}_{t,i}^{R}\right)$
  $\mathcal{H}_t \leftarrow \mathcal{H}_{t-1} + \textsc{ProjectResidual}(\mathcal{Q}_t, \overline{\mathbf{B}}_R, m_t) + \Delta\mathbf{a}_t^{R,\text{appl}}$
  $r_{t,R} \leftarrow \rho_R(\mathcal{H}_t)$, $r_{t,T} \leftarrow 0$
  $\widehat{\mathbf{s}}_{t+1} \leftarrow \Psi(\mathcal{F}_t, \mathbf{A}_t)$
  $t \leftarrow t + 1$
end while

computation, and packing. Normalized values remain in registers, and the kernel writes packed activations and their scales for the subsequent GEMM.

GEMM epilogues preserve the model's operation order and BF16 rounding boundaries. The QKV epilogue applies RoPE to Q and K and writes K/V directly to the cache. The GateUp epilogue combines the model's tanh-based GELU with multiplication, producing a compact product instead of separate Gate and Up tensors. O and Down epilogues combine scaling, gating, and residual addition into the final output store. Where supported, Down quantizes its input during tile loading, eliminating an additional packed-activation write and read.

We specialize scheduling for the different Prefix and Flow matrix shapes. Asynchronous copies and multiple shared-memory stages overlap tile loading with computation. For the small-row Flow Down projection, split-K execution increases parallelism, with INT32 partial sums stored in a preallocated workspace and reduced in a fixed order before output processing.

For the fixed $\pi_{0.5}$ deployment, a Prefix graph and a Flow graph containing ten unrolled flow steps execute through CUDA Graph replay. Inputs, caches, outputs, and workspaces retain fixed addresses. Precision paths are prepared before execution, removing allocation, weight repacking, compilation, and backend search from replay.

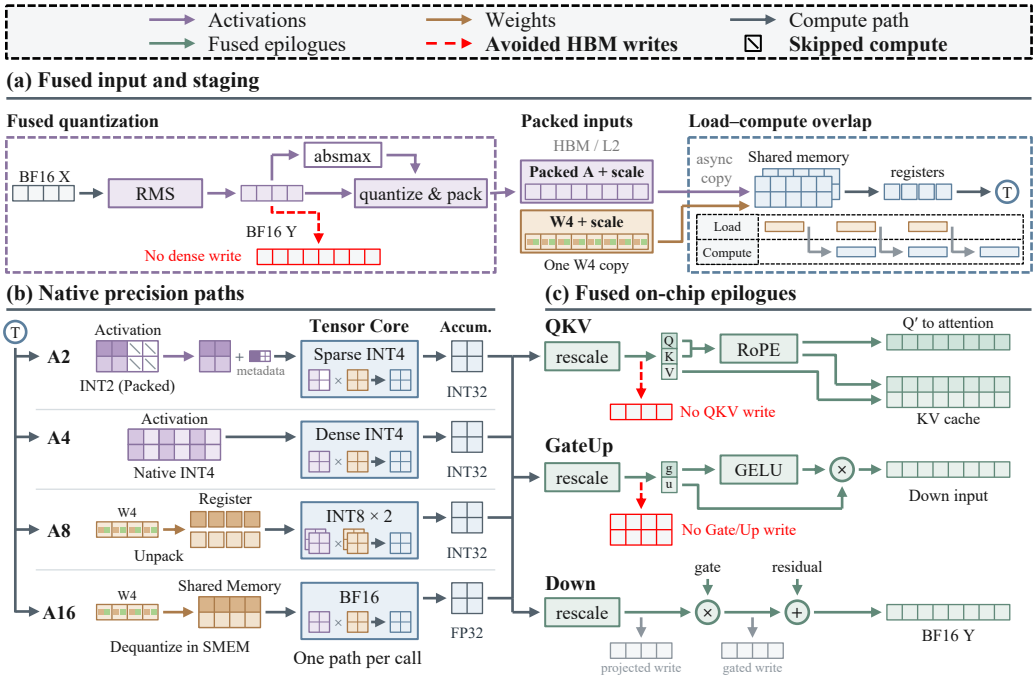

Figure 12: **Implementation details of the Proposed *DyQ-VLA* Framework.**

E Extended Experiments

E.1 Experimental Setup

Models and Benchmarks. We evaluate $\pi_{0.5}$ (Physical Intelligence et al., 2025) and GR00T-N1.5 (Bjorck et al., 2025) on the Spatial, Object, Goal, and Long suites of LIBERO (Liu et al., 2023). We additionally evaluate $\pi_{0.5}$ on RoboTwin 2.0 (Chen et al., 2025) under Clean and Randomized settings. We use official checkpoints and benchmark protocols, with 50 evaluation rollouts per LIBERO task and 1000 rollouts per suite. Inference runs on an NVIDIA A100 GPU.

Baselines and Calibration. BF16 serves as the full-precision reference. On LIBERO, we compare five general-purpose post-training quantization methods (Frantar et al., 2022; Lin et al., 2024b; Shao et al., 2024; Xiao et al., 2023; Lin et al., 2024a) and four VLA-specific methods (Akbari et al., 2026; Xu et al., 2026c; Zhang et al., 2026a; Wang et al., 2026c); the latter are also evaluated on RoboTwin 2.0. All quantized methods share 60 LIBERO calibration trajectories, which are used to configure quantization rather than counted as evaluation rollouts. Baselines retain their original configurations, while *DyQ-VLA* uses 4-bit weights and selects activation precision from $\{2, 4, 8, 16\}$ bits during execution.

E.2 Evaluation Metrics

Task Success and Execution Steps. For an evaluation set containing N rollouts, let $s_i \in \{0, 1\}$ indicate whether rollout i succeeds and let K_i denote its execution-step count. Success Rate (SR) includes all rollouts, whereas the reported execution steps average only successful episodes, as shown in Eq. (17). Failed rollouts therefore reduce SR but do not contribute to the reported mean completion steps. Since each LIBERO suite contains the same number of evaluation rollouts, the four-suite mean SR also equals the success rate pooled across the four suites. Overall execution steps are weighted by the number of successful episodes in each suite, rather than averaged equally across suite-level means.

$$\mathrm{SR} = \frac{100}{N}\sum_{i=1}^{N} s_i, \qquad \overline{K} = \frac{\sum_{i=1}^{N} s_i K_i}{\sum_{i=1}^{N} s_i}. \tag{17}$$

Runtime Errors. We measure end-effector deviations from matched and aligned demonstration trajectories after each executed action chunk. Translation error is the Euclidean distance between the executed and reference positions, and rotation error is the shortest angular distance between their orientations. The reported units are millimeters and degrees, respectively. Only successful episodes contribute to the task-evaluation error statistics, so the reported errors characterize deviations during task completion and are interpreted together with SR. For LIBERO, the overall error is the arithmetic mean of the four suite-level error means, giving each suite equal weight. The executed chunk contains five actions for $\pi_{0.5}$ and one action for GR00T-N1.5 on LIBERO. These execution checkpoints are distinct from the internal integration steps used to generate actions.

Inference Efficiency. Step speedup measures model inference time per policy query, while Rollout speedup measures cumulative model inference time over an episode. If an episode contains Q policy queries with inference times $\ell_1, \ldots, \ell_Q$, its cumulative inference time is $\sum_{t=1}^{Q} \ell_t$. Both speedups normalize the corresponding BF16 inference time by that of the evaluated method, so a value greater than one indicates faster inference. Rollout speedup captures the combined effects of query latency and the number of policy queries required during execution. Consequently, a method with faster individual queries can still have a rollout speedup below one if task execution requires sufficiently more queries. The memory values in the LIBERO efficiency comparison represent static full-model weight footprints in GB.

E.3 Runtime Errors Across LIBERO Suites

Suite-Level Results. Tab. 4 reports SR, translation error, rotation error, and execution steps for $\pi_{0.5}$ across all four LIBERO suites. Among the methods listed, *DyQ-VLA* achieves the lowest translation and rotation errors and the fewest execution steps in each suite. On LIBERO-Long, both *DyQ-VLA* and QuantVLA achieve 96.0% SR, while translation error decreases from 91.98 to 62.45 mm, rotation error from 26.60 to 14.04 degrees, and mean execution steps from 224.5 to 163.3. Thus, the same final success rate can accompany substantially different trajectory deviations and execution lengths.

Runtime Error Distributions. Fig. 13 compares the translation and rotation error distributions on LIBERO-Long. *DyQ-VLA* exhibits a more concentrated translation-error distribution and a rotation-error distribution centered at smaller values. These distributions complement the suite-level averages by showing the dispersion and location of the observed errors.

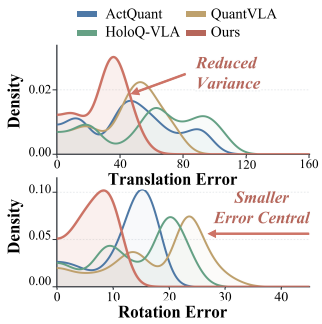

Figure 13: Error Densities.

F Real-World Evaluation Details

We evaluate *DyQ-VLA* on real-world tabletop manipulation tasks to examine whether its inference efficiency translates into faster task completion while preserving task performance. This section describes the robotic platform, task coverage, policy configurations, and evaluation protocol.

F.1 Robotic Platform And Workspace

Hardware Setup. The experiments use an Agilex Piper-X robotic arm equipped with a one-degree-of-freedom gripper and an Orbbec Dabai camera. Fig. 14 (a) identifies the arm, gripper, camera, and camera holder in the tabletop workspace. The manipulation objects include fruit models, plates, cups, and a case with a lid. These objects support both individual placement and insertion operations and tasks that combine several successive operations.

F.2 Task Design

Evaluation Setting. Existing robotic resources support different aspects of policy learning and evaluation. Open X-Embodiment (Open X-Embodiment Collaboration, 2024) provides demonstrations from multiple robot embodiments, RoboCasa (Nasiriany et al., 2024) provides simulated kitchen

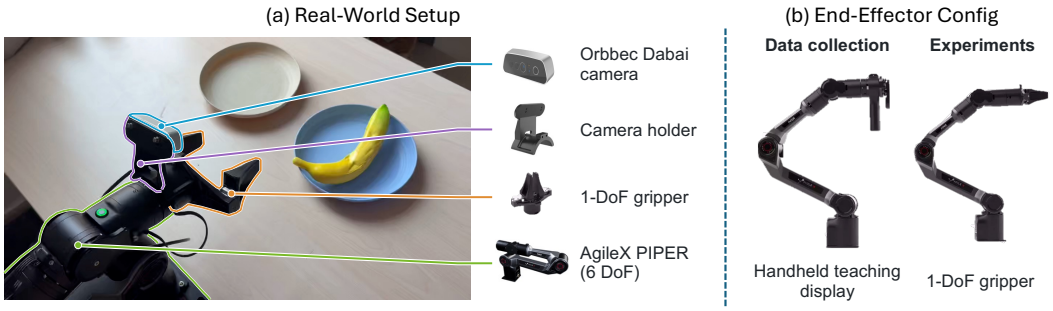

Figure 14: **Real-World Platform And End-Effector Configurations.** (a) Tabletop Workspace And Hardware Components. (b) Configurations for Demonstration Collection And Task Execution.

Table 6: Overview of Robotic Data And Evaluation Settings.

Resource	Setting	Scope	Primary Role
OXE	Real-world	Demonstrations across multiple robot embodiments	Robot learning data
RoboCasa	Simulation	Household manipulation in kitchen environments	Task environments and demonstration data
SimplerEnv	Simulation	Manipulation setups corresponding to real robots	Policy evaluation in simulation
Our setup	Real-world	Six tabletop tasks on a single robotic platform	Task success and completion-time evaluation

manipulation environments and demonstration data, and SimplerEnv (Li et al., 2024) evaluates policies in simulation setups corresponding to real robots. Tab. 6 summarizes these roles. Our tabletop platform supports direct evaluation of task success and task completion time during physical execution.

Task Coverage. We consider four short-horizon tasks and two multi-stage tasks. The short-horizon tasks involve placing a banana or an apple on a plate, inserting one cup into another, and inserting a banana into a cup. The multi-stage tasks extend this coverage to orange packing followed by lid closure and placement of two objects in a specified order. Tab. 7 lists the objective of each task, and Fig. 15 illustrates representative execution sequences.

F.3 Data Collection And Policy Setup

Teaching And Execution Configurations. Fig. 14 (b) distinguishes the two end-effector configurations used for data collection and experiments. The handheld teaching configuration provides an interface for manually guided demonstrations, whereas the gripper configuration supports object manipulation during task execution. This distinction separates demonstration acquisition from the autonomous executions used to evaluate the policy.

Policy Backbone. The real-world evaluation uses $\pi_{0.5}$ (Physical Intelligence et al., 2025), with its BF16 execution serving as the reference for task performance and completion time. All comparisons use this backbone under the precision configurations described below.

F.4 Quantization And Online Execution

Precision Configurations. We compare *DyQ-VLA* with QuantVLA (Zhang et al., 2026a), ActQuant (Akbari et al., 2026), QVLA (Xu et al., 2026c), and HoloQ-VLA (Wang et al., 2026c). Their evaluated configurations are W4A8, W4A16, W4A16, and W4A4, respectively, where W and A denote weight and activation precision. *DyQ-VLA* maintains 4-bit weights and selects activation

Table 7: Real-World Tasks And Objectives.

Task Group	Task	Objective
Short-horizon	Banana plate	Place the banana on the beige plate.
	Cup insertion	Insert one cup into the other cup.
	Banana cup	Insert the banana into the cup.
	Apple plate	Place the apple on the plate.
Multi-stage	Orange packing	Place the orange in the case and close the lid.
	Ordered placement	First place the apple on the beige plate, then place the banana on the blue plate.

*Note: Each method is evaluated in ten trials per task. Performance results are reported in Tab. 3.

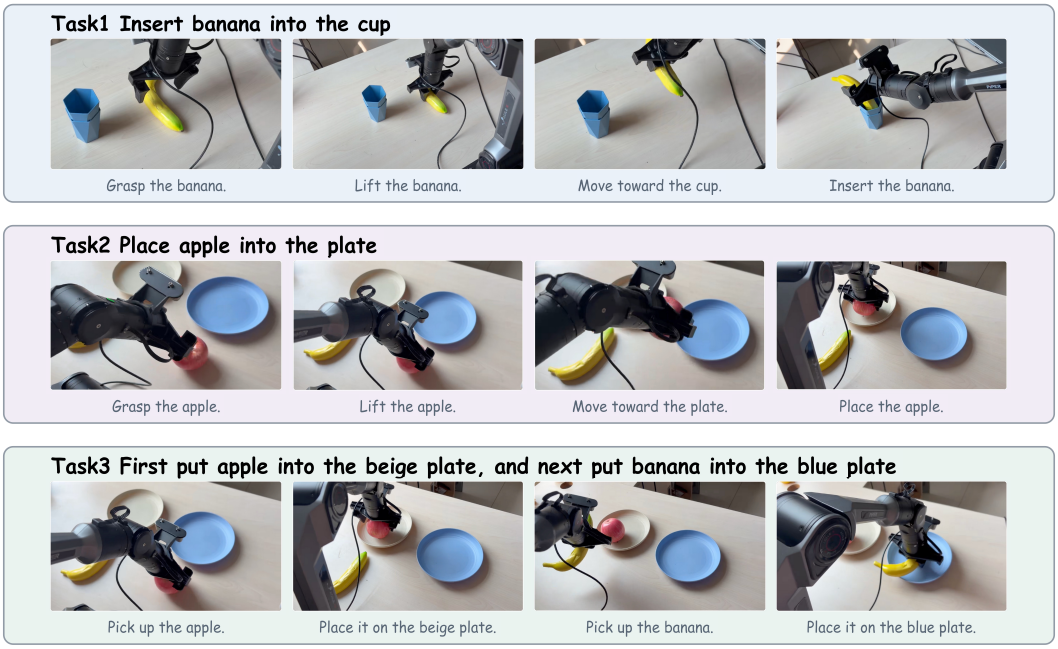

Figure 15: **Representative Real-World Task Sequences.** The examples show banana insertion, apple placement, and ordered placement of the apple and banana on their respective plates.

precision from $\{2, 4, 8, 16\}$ bits during execution. Thus, the comparisons cover fixed precision configurations and the dynamic activation precision used by *DyQ-VLA*.

Calibration Protocol. All quantized methods use the same 60 LIBERO calibration trajectories. For *DyQ-VLA*, offline calibration records the error levels and inference latency of the candidate activation configurations. These statistics support precision selection through table lookup and constraint evaluation during execution.

Runtime Error Control. Before each policy query, *DyQ-VLA* uses estimated response sensitivity and historical rotational residuals to select the lowest-latency configuration satisfying the error constraints. Selection uses precomputed calibration statistics rather than running every candidate configuration online. The rotational components of the actions to be executed are then adjusted using residual-based compensation. Precision-specific operators and fused execution support this dynamic inference path. Together, these components connect numerical precision selection with action generation and compensation during execution.

F.5 Evaluation Protocol

Trials And Task Success. Each method is evaluated in ten trials per task, giving 60 trials across the six tasks. For each task, we report the successful-trial count out of ten, and compute the overall success rate by dividing the total number of successful trials by 60. Because every task has the same number of trials, this overall rate also equals the arithmetic mean of the six task-level success rates. Both successful and unsuccessful trials contribute to this metric; the task objectives and their grouping are listed in Tab. 7. *DyQ-VLA* succeeds in 44 of 60 trials (73.3%), compared with 45 of 60 trials (75.0%) for BF16.

Task Completion Time. We report end-to-end task completion speedup relative to BF16 as Task Spd. This metric measures the time required for physical task completion. Step and Rollout speedups instead measure model inference time per query and cumulative model inference time per episode, respectively. *DyQ-VLA* achieves a $1.59\times$ task completion speedup over BF16, compared with $1.16\times$ for QuantVLA, the fastest quantized baseline.